Web Crawling with Python

This is a brief guide to web crawling with python. By “web crawling,” we mean programatically traversing and parsing web pages for data using robots; (naturally) we call these robots “spiders.”

This guide will assume basic knowledge of python, and will use python 2.7. (It shouldn’t be hard to adapt to python 3.) It’s slightly adapted from the guide here, using BeautifulSoup instead of building our own parser.

Tools Needed

`requests`

`requests` is a very useful library that simplifies making requests to web pages. You need to know a litle about the web to understand what “requests” are, but the long story short is that it’s a “request” to load a web page (like you do every time you visit a url!). There are other ways and modules to access urls (historically, `urllib` is common), but when you get to APIs–as mentioned at the end of this article– `requests` will make your life much easier.

`BeautifulSoup`

BeautifulSoup is a fantastically useful python library that will “parse” web pages for you into easily accessible python data. Specifically, it

parses HTML (Hypertext Markup Language, the fundamental code in which **all** web pages are writen). You don't need to know HTML here, but here's an example so we're all on the same page:

```
<h1>This is a header tag.</h1>
<p>This is a paragraph tag. I can make things <b>bold</b>
 or create links <a href="http://google.com">like this on
e</a> by embedding things in more nested levels of tags.<
/p>
```

This is a massive oversimplification of HTML, but demonstrative of the general structure. Using BeautifulSoup (BS) we can turn this HTML into some nice objects so we don't have to do things like turn this HTML into a string and manually search through it.
Learning to web program is beyond the scope of this article, but it's a massive, massive world.

Getting our feet wet

The best way to learn is by doing, so let's start!

```
# import our modules
import requests
from bs4 import BeautifulSoup as bs
```

Now that we have our modules, let's pick a website to crawl. Since this is only a simple introductory article, we're going to restrict ourselves

to a very small website with few pages and links: the CS 134 website.

```
# set our root url
root_url = "http://harvardcs134.com"
```

Now, for our first interaction with `requests` and `BeautifulSoup`. We're going to request the web page, and then load it into a BS object.

```
# request page using a "GET" request
r = requests.get(root_url)
# actually get the text of the page (in this case, HTML!)
page_text = r.text
# store the HTML in a BS object; we'll use the "lxml" par
sing system
soup = bs(page_text, "lxml")
# we can actually see our web page stored in the BS objec
t using one of the BS methods
# for example, here's a sample 400 characters...
print soup.prettify()[7600:8000]
```

```
u-item-home current-menu-item page_item page-item-4 curre
nt_page_item menu-item-46" id="menu-item-46">
          <a href="http://www.harvardcs134.com/">
           Announcements
          </a>
         </li>
         <li class="menu-item menu-item-type-post_type men
```

```
u-item-object-page menu-item-51" id="menu-item-51">
          <a href="http://www.harvardcs134.com/index.php/s
yllabus/">
           Syllabus
```

We want to “crawl” the website, so we want to find all the links on this page so we can follow them later.

To do this, we can use BS! BS’s `find` and `find_all` methods will take a “selector” and return BS objects representing these “objects” in the web page. For example, if we supply the `'a'` selector, BS will look for all anchor (`<a>`) tags, or the HTML tag type that represents links. (`find` gives the first, `find_all` gives all on the page.) There are many different possible selectors, but we’ll keep it simple. These tags will be stored in another BS object we can use.

```
links = soup.find_all('a')
print "There are %d links on this page" % len(links)
print "They're now stored in a %s object" % type(links)
```

```
There are 9 links on this page
They're now stored in a <class 'bs4.element.ResultSet'> o
bject
```

And now we just have to actually retrieve the links, which we can do by accessing the `href` attribute.

```
print "Here are the links on the page: "
print [l.get('href') for l in links]
```

```
Here are the links on the page:
['http://www.harvardcs134.com/', '#content', 'http://www.
harvardcs134.com/', 'http://www.harvardcs134.com/index.ph
p/syllabus/', 'http://www.harvardcs134.com/index.php/lear
ning-objectives/', 'http://www.harvardcs134.com/index.php
/assignments/', 'http://www.harvardcs134.com/index.php/co
urse-materials/', 'http://www.harvardcs134.com/index.php/
course-resources/', 'https://wordpress.org/']
```

Building our crawler

We have all the tools we need, so let's formalize all this and build a class that will take care of all this for us! We'll call our class `Spider134`. It'll optionally take a maximum number of links to follow (so we don't spiral out of control) and a domain to stick to (same idea, so that we can stick to the 134 website). We'll give it one main method that will crawl for links. While crawling, we'll store the title to demonstrate some more BeautifulSoup powers, and build a web graph, since this is a networks class, after all.

```
class Spider134():

    def __init__(self, maxNumLinks=20, domain='http://www
.harvardcs134.com'):
```

```
        self.maxNumLinks = maxNumLinks
        self.domain = domain

    def crawl(self, urls):
        numCrawled = 0
        linksToCrawl = list(urls)
        # we'll keep a set of links we've actually traver
sed, where the link maps to its title
        visitedLinks = {}
        # same, but now for edges to other links
        graph = dict()
        while len(linksToCrawl) > 0 and numCrawled < self
.maxNumLinks:
            link = linksToCrawl.pop()
            # don't loop forever!
            if link in visitedLinks:
                continue
            print "Now crawling...", link
            # note that through all of this I'm assuming
no exceptions will
            # be thrown because of bad links or such; tha
t's a bad assumption
            # --you should use a try-catch in your code!!

            r = requests.get(link)
            # we're also going to restrict ourselves a li
ttle by only exploring HTML
            # there are other types that are interesting,
```

```
 but those are for another day
            if r.headers['Content-Type'].find('text/html'
) == -1:
                continue
            soup = bs(r.text, "lxml")
            # increment our visited properties
            numCrawled += 1
            visitedLinks[link] = soup.title.string
            # make sure not to double-count links
            to_links = list(set([l.get('href') for l in s
oup.find_all('a')]))
            to_links = [l for l in to_links if l.find(sel
f.domain) == 0]
            graph[link] = to_links
            linksToCrawl += to_links
        # do some final clean up to remove non-html edges
 in graph
        for u in graph:
            for v in graph[u]:
                if v not in graph:
                    graph[u].remove(v)
        # and store our data
        self.graph = graph
        self.links = visitedLinks
```

```
spider = Spider134()
spider.crawl(['http://harvardcs134.com'])
```

```
import pprint
pp = pprint.PrettyPrinter(indent=4)
print "Here our links and their titles: "
pp.pprint(spider.links)
print "And our graph of edges: "
pp.pprint(spider.graph)
```

```
Now crawling... http://harvardcs134.com
Now crawling... http://www.harvardcs134.com/index.php/cou
rse-materials/
Now crawling... http://www.harvardcs134.com/index.php/ass
ignments/
Now crawling... http://www.harvardcs134.com/index.php/lea
rning-objectives/
Now crawling... http://www.harvardcs134.com/
Now crawling... http://www.harvardcs134.com/index.php/syl
labus/
Now crawling... http://www.harvardcs134.com/index.php/cou
rse-resources/
Here our links and their titles:
{   'http://harvardcs134.com': u'CS134 : NETWORKS',
    'http://www.harvardcs134.com/': u'CS134 : NETWORKS',
    'http://www.harvardcs134.com/index.php/assignments/':
 u'Assignments | CS134 : NETWORKS',
    'http://www.harvardcs134.com/index.php/course-materia
ls/': u'Course Materials | CS134 : NETWORKS',
```

```
    'http://www.harvardcs134.com/index.php/course-resourc
es/': u'Resources | CS134 : NETWORKS',
    'http://www.harvardcs134.com/index.php/learning-objec
tives/': u'Learning Objectives | CS134 : NETWORKS',
    'http://www.harvardcs134.com/index.php/syllabus/': u'
Syllabus | CS134 : NETWORKS'}
And our graph of edges:
{   'http://harvardcs134.com': [   'http://www.harvardcs1
34.com/index.php/learning-objectives/',
                                   'http://www.harvardcs1
34.com/index.php/course-resources/',
                                   'http://www.harvardcs1
34.com/index.php/syllabus/',
                                   'http://www.harvardcs1
34.com/',
                                   'http://www.harvardcs1
34.com/index.php/assignments/',
                                   'http://www.harvardcs1
34.com/index.php/course-materials/'],
    'http://www.harvardcs134.com/': [   'http://www.harva
rdcs134.com/index.php/learning-objectives/',
                                        'http://www.harva
rdcs134.com/index.php/course-resources/',
                                        'http://www.harva
rdcs134.com/index.php/syllabus/',
                                        'http://www.harva
rdcs134.com/',
                                        'http://www.harva
```

```
rdcs134.com/index.php/assignments/',
                                        'http://www.harva
rdcs134.com/index.php/course-materials/'],
    'http://www.harvardcs134.com/index.php/assignments/':
 [   'http://www.harvardcs134.com/index.php/syllabus/',

     'http://www.harvardcs134.com/index.php/course-resour
ces/',

     'http://www.harvardcs134.com/',

     'http://www.harvardcs134.com/index.php/learning-obje
ctives/',

     'http://www.harvardcs134.com/index.php/assignments/'
,

     'http://www.harvardcs134.com/index.php/course-materi
als/'],
    'http://www.harvardcs134.com/index.php/course-materia
ls/': [   'http://www.harvardcs134.com/index.php/learning
-objectives/',

          'http://www.harvardcs134.com/index.php/course-r
esources/',

          'http://www.harvardcs134.com/index.php/syllabus
/',
```

```
          'http://www.harvardcs134.com/',

          'http://www.harvardcs134.com/index.php/assignments/',

          'http://www.harvardcs134.com/index.php/course-materials/'],
    'http://www.harvardcs134.com/index.php/course-resources/': [   'http://www.harvardcs134.com/index.php/learning-objectives/',

          'http://www.harvardcs134.com/index.php/course-resources/',

          'http://www.harvardcs134.com/index.php/syllabus/',

          'http://www.harvardcs134.com/',

          'http://www.harvardcs134.com/index.php/assignments/',

          'http://www.harvardcs134.com/index.php/course-materials/'],
    'http://www.harvardcs134.com/index.php/learning-objectives/': [   'http://www.harvardcs134.com/index.php/learning-objectives/',
```

```
            'http://www.harvardcs134.com/index.php/course-resources/',

            'http://www.harvardcs134.com/index.php/syllabus/',

            'http://www.harvardcs134.com/',

            'http://www.harvardcs134.com/index.php/assignments/',

            'http://www.harvardcs134.com/index.php/course-materials/'],
    'http://www.harvardcs134.com/index.php/syllabus/': [
  'http://www.harvardcs134.com/index.php/learning-objectives/',

  'http://www.harvardcs134.com/index.php/course-resources/',

  'http://www.harvardcs134.com/index.php/syllabus/',

  'http://www.harvardcs134.com/',

  'http://www.harvardcs134.com/index.php/assignments/',

  'http://www.harvardcs134.com/index.php/course-materials
```

```
/']}
```

Comments on Our Spider

This spider isn't super efficient: I do a DFS crawl, rather than a BFS, and I rather naively add and explore edges. You can tell that it's not intelligent either: it can't recognize 'http://harvardcs134.com' and 'http://www.harvardcs134.com' as the same, and so adds and explores them twice. That's okay! This is the kind of basic link parsing that you can expand upon whenever you build your own spider (or let prior developed spiders do for you, as we'll soon see).

We don't particularly stress test this spider, only running it on the tiny cs134 website. On larger websites, whatever spider you will have to have time and memory considerations, and those parameters for how to restrict the number of crawls and your domain will come to matter significantly.

We also do no more than store our data as another python object. This can be very unwieldy, so a better approach would be to write it as JSON to a file, or to clean the data further after crawling.

Moving forward

BeautifulSoup

There are lots and lots of cool things you can do with BeautifulSoup!

We've only skimmed the surface by looking for links and the title. Data scientists especially like BeautifulSoup when websites have data listed very repetitively in the same format over and over (like in tables); BS usually makes quick work of these. As an example, the Harvard Facebook stores its user data in tables…remember the scene in *The Social Network* when Zuckerberg parses all the student pictures from the house websites? Then and now, it's basically some quick BS.

Scrapy

Actually, there's already been a package built that combines these link crawling features with BS's parsing abilities. It's called scrapy.

This guide won't explore scrapy, but you're encouraged to explore it on your own. It has its own tutorial here, with a slightly more user friendly version here. The usage is fundamentally similar to BeautifulSoup and requests.

APIs

Actually (x2), these days a lot of services won't force you to scrape their websites: they'll offer it up in an API, or "Application Programming Interface," which is basically a bunch of pre-specified links that explicitly return data in an already machine-readable form, rather than requiring you to parse it from HTML. Many big data companies like Google and Facebook already have public APIs with well written documentation. Remember the `request` library? It can be incredibly helpful in accessing the…less well documented APIs, when

you have to sometimes guess at the urls. (For example: Tinder has no publicly released API, but people have already crawled all the API urls through trial and error.)

Be ethical!

Many websites don’t want you scraping their data! Either because you’re supposed to view the data naturally (for ad revenue, for example); because their data is their entire service (as happened with the woman who scraped Elsevier’s research archives then gave it away for free); because of the stress on their servers (these spiders are just additional web requests, after all); or for many other reasons.

Indeed, Google’s own web crawlers, which scour the web to find new links, will respect files listed on these websites called `robot.txt`'s which the owners might use to tell Google not to crawl that website, or only crawl in a specific way. You should respect these policies as well!